DW_asymdata_inbuf

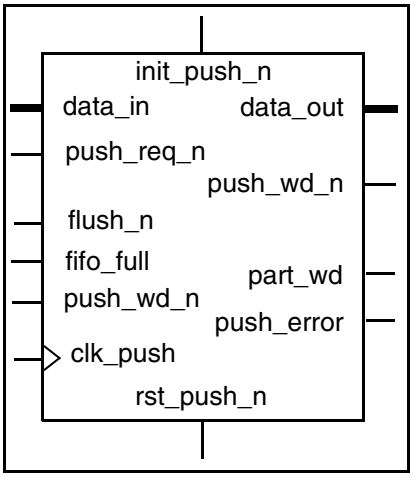
Asymmetric Data Input Buffer

Version, STAR, and myDesignWare Subscriptions: IP Directory

Features and Benefits

Revision History

- Parameterized asymmetric input and output data widths (*in_width* < *out_width* with integer multiple relationship)
- Parameterized byte (sub-word) ordering within a word
- Parameterized *flush_value*
- Word integrity flag
- Parameterized error status mode
- Registered push error (overflow)



Description

This component buffers up input data stream sub-words and assembles them into output words predicated on the input data width being less than and an integer multiple of the output data width. Once the number of input sub-words received completes a full data word, the data output is presented along with a push word enable signal.

This component was initially conceived as a back-end piece to an asymmetric First-In-First-Out (FIFO) device as depicted in the data flow usage example shown in Figure 1-2 on page 5. Note that the interface naming of DW_asymdata_outbuf incorporates the "push" nomenclature associated with a FIFO device.

A "flush" feature is available to force out a partially buffered word. This functionality is useful in clearing the input buffers and establishing word alignment.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk_push	1	Input	Clock source
rst_push_n	1	Input	Asynchronous reset (active low)
init_push_n	1	Input	Synchronous reset (active low)
push_req_n	1	Input	Push request (active low)
data_in	*in_width*	Input	Input data (word)
flush_n	1	Input	Flush the partial word (active low)
fifo_full	1	Input	Full indication connected RAM/FIFO
push_wd_n	1	Output	Ready to write full data word (active low)

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
data_out	*out_width*	Output	Output data (sub-word)
inbuf_full	1	Output	Input registers all contain active `data_in` sub-words
part_wd	1	Output	Partial word pushed flag
push_error	1	Output	Overrun of RAM/FIFO (includes input registers)

Table 1-2 Parameter Description

Parameter	Values	Description
in_width	1 to 2048 Default: 8	Width of `data_in` Must be less than *out_width* and an integer multiple; that is, *out_width* = K * *in_width*
out_width	1 to 2048 Default: 16	Width of `data_out` Must be greater than *in_width* and an integer multiple; that is, *out_width* = K * *in_width*
err_mode	0 or 1 Default: 0	Error flag behavior mode ■ 0: Sticky `push_error` flag (hold on first occurrence, clears only on reset) ■ 1: Dynamic `push_error` flag (reports every occurrence of error)
byte_order	0 or 1 Default: 0	Sub-word ordering into Word ■ 0: The first byte (or sub-word) is in MSB of word ■ 1: The first byte (or sub-word) is in LSB of word
flush_value	0 or 1 Default: 0	Fixed flushing value ■ 0: Fill empty bits of partial word with 0's upon flush 1: Fill empty bits of partial word with 1's upon flush

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis model	DesignWare

Table 1-4 Simulation Models

Model	Function
DW03.DW_ASYMDATA_INBUF_CFG_SIM	Design unit name for VHDL simulation
dw/dw03/src/DW_asymdata_inbuf_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_asymdata_inbuf.v	Verilog simulation model source code

Table 1-5 lists all the combinations of `flush_n` and `push_req_n` in relation to `part_wd`, `inbuf_full`, and `fifo_full` and the resulting action that is performed.

Table 1-5 Flush and Push scenarios and results

flush_n	push_req_n	part_wd	inbuf_full	fifo_full	Result/Action
1	0	x	0	x	push only, no error
0	0	0	0	x	push only, no error
1	0	1	1	0	generate `push_wd_n`, reset counters, no error
x	0	1	1	1	push error, last subword lost, hold counters
0	1	1	x	0	flush only, reset counters, no error *
0	1	0	0	x	do nothing
0	0	1	x	0	flush and push (`data_in` into sub-word1 input reg.), no error *
0	1	1	x	1	push error, no other action
0	0	1	0	1	push error, no flush, push data
1	1	x	x	x	do nothing
x	x	0	1	x	not possible
* During a valid flush condition, `push_wd_n` is asserted					

Figure 1-1 is a block diagram of the DW_asymdata_inbuf component.

Figure 1-1 DW_asymdata_inbuf Basic Block Diagram

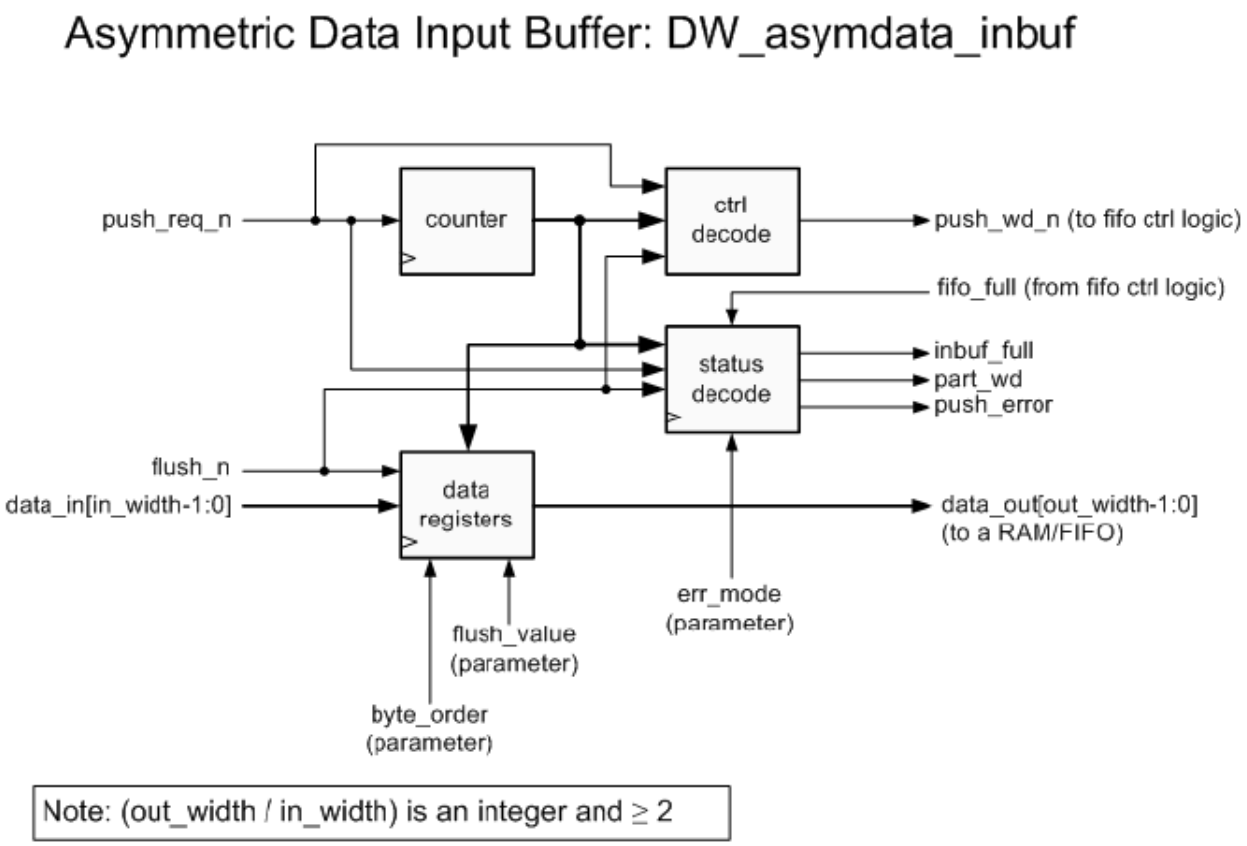
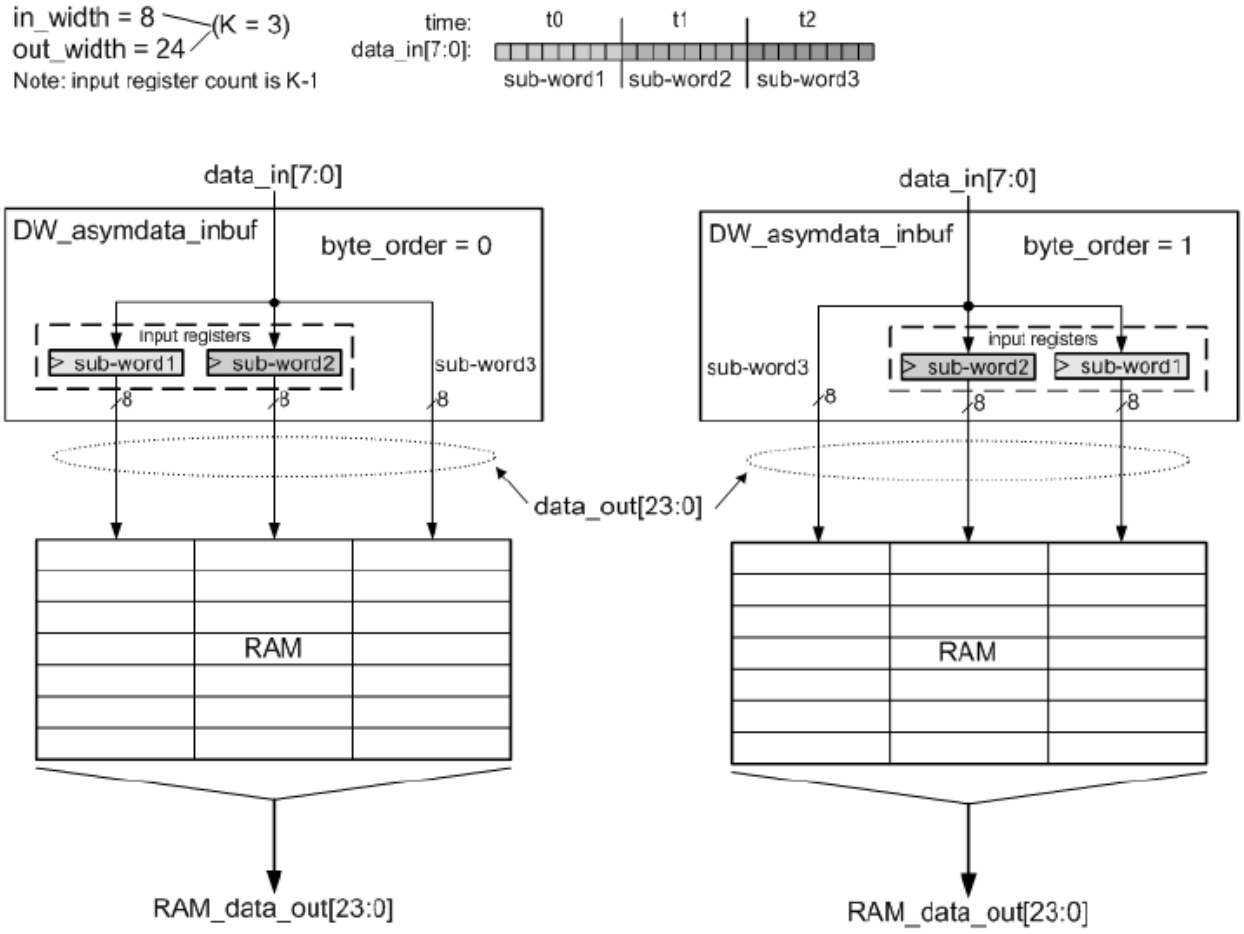


Figure 1-2 shows how the input registers are organized and routed to `data_out` (based on the value of *byte_order*) to a RAM element. Note that the number of sub-word input registers is equal to (*out_width* / *in_width*) - 1.

Figure 1-2 Data Flow Based on 'byte_order' Value



Partial Words

When a partial word is in the input registers, output flag `part_wd` is active (high). After K pushes, where K = *out_width* / *in_width*, K sub-words are assembled into a full word (K-1 sub-words in the input register and the last sub-word on the `data_in` bus). This full word is then presented for writing into RAM/FIFO. When a full word is sent from the input registers to RAM/FIFO, `part_wd` goes inactive (low) on the following cycle. The order of sub-words within a word is determined by the *byte_order* parameter (as shown in Figure 1-2).

Pushing Complete Words

As the Kth sub-word is being pushed (`push_req_n` asserted low), the complete word is presented to the `data_out` output along with assertion of `push_wd_n` (low single `clk_push` cycle). The `push_wd_n` output is non-registered single-clock pulse and is a combinational result derived directly from `push_req_n`. So, there must be an awareness of the timing characteristics of `push_req_n`.

Flushing Input Registers

The flushing function, via `flush_n` assertion, pushes a partial word to RAM/FIFO. The input registers are pre-set to the *flush_value* value after a flush with the exception of one case. This case is when simultaneous assertion of `flush_n` and `push_req_n` when `part_wd` is asserted and `fifo_full` is not asserted. Under this scenario, the sub-word from `data_in` is placed into the sub-word1 input register with all other sub-word locations pre-set the *flush_value* value.

A complete list of results caused by `flush_n` assertion is shown in Table 1-5 on page 3.

Push Errors

A push error occurs under either of these conditions:

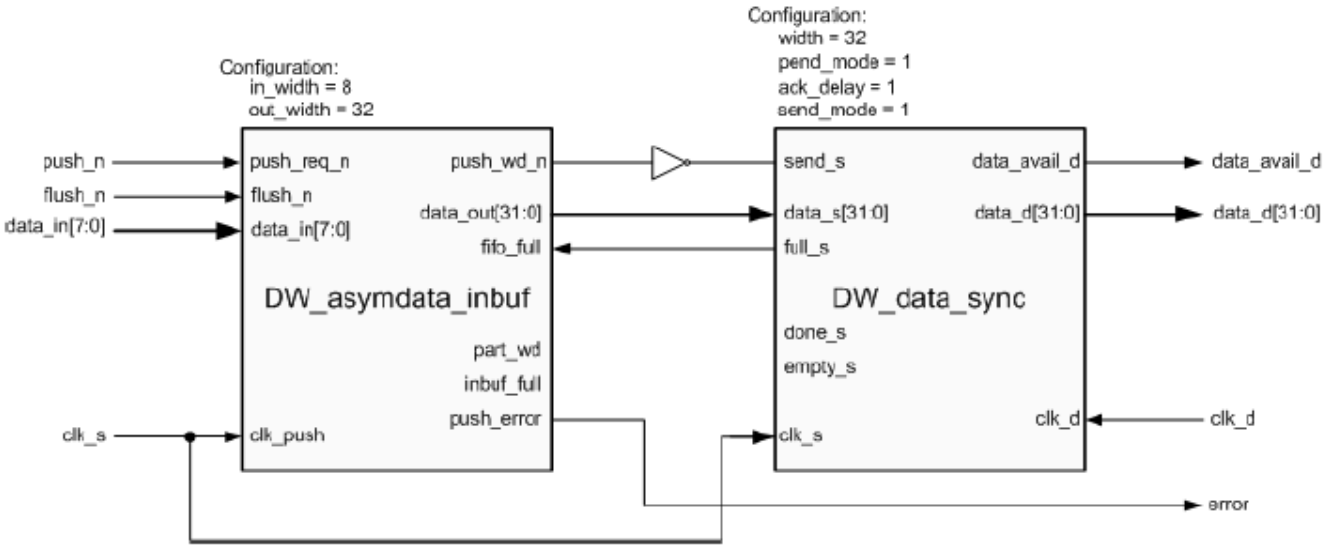
- `push_req_n` active, `part_wd` active, `inbuf_full` active, and `fifo_full` active OR
- `flush_n` active, `part_wd` active, and `fifo_full` active

The `push_error` output is registered and its behavior can either be 'sticky' or 'dynamic' based on the *err_mode* parameter. For the definition of *err_mode*, see Table 1-2 on page 2.

Example of non-FIFO Use

Figure 1-3 depicts an example of the DW_asymdata_inbuf connected to a DesignWare synchronizing component called the DW_data_sync. The DW_data_sync synchronizes incoming data from one clock domain to another. Here, the DW_asymdata_inbuf provides the data from the source domain and the DW_data_sync which then passes it on to the destination domain.

Figure 1-3 Example of DW_asymdata_inbuf Used with DW_data_sync



Suppressing Warning Messages During Verilog Simulation

The Verilog simulation model includes macros that allow you to suppress warning messages during simulation.

To suppress all warning messages for all DWBB components, define the DW_SUPPRESS_WARN macro in either of the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

  ```
  `define DW_SUPPRESS_WARN
  ```

- Or, include a command line option to the simulator, such as:

  `+define+DW_SUPPRESS_WARN` (which is used for the Synopsys VCS simulator)

The warning messages for this model include the following:

- If values other than 1 or 0 are present on a clock port, the following message is displayed:

  ```
  WARNING: <instance_path>.<clock_name>_monitor:
      at time = <timestamp>, Detected unknown value, x, on <clock_name> input.
  ```

  To suppress only this warning message for all DWBB components, use the following macro:

  - Define the DW_DISABLE_CLK_MONITOR macro. You can define this macro in the following ways:
    - Specify the Verilog preprocessing macro in Verilog code:

      ```
      `define DW_DISABLE_CLK_MONITOR
      ```

    - Or, include a command line option to the simulator, such as:

      `+define+DW_DISABLE_CLK_MONITOR` (which is used for the Synopsys VCS simulator)

  This message is also suppressed using the DW_SUPPRESS_WARN macro explained earlier.

Timing Waveforms

Figure 1-4 depicts, after an asynchronous reset, a sequence of pushing a complete word followed by simultaneous push-flush assertion while `part_wd` is active. The data widths configurations are *in_width* of 4 and *out_width* of 16 which makes the integer multiple of *in_width* to *out_width*, K, equal to 4. This implies that the number of input sub-word registers is 3 (K-1). So, after three pushes `inbuf_full` goes active. On the fourth active `push_req_n`, `push_wd_n` goes active (low) which implies valid `data_out`. Subsequently, the `inbuf_full` output goes low on the next cycle after the `push_wd_n` active pulse. The *byte_order* parameter in this case is set to 0 which means that the first sub-word pushed is placed to the most significant sub-word of `data_out`.

Following the pushing of a complete word, two pushes (causes `part_wd` to go active) occur then a simultaneous push and flush event. The flush is acted upon because `fifo_full` is not asserted and `part_wd` is asserted. So, the result of this simultaneous push-flush causes whatever is already in the input registers to be written to `data_out` and any non pushed sub-words written to `data_out` with a value based on the *flush_value* parameter setting. In this case, *flush_value* is set to 1. So, the resulting value of `data_out` is '0xd5ff' where the 'd' and '5' were already stored in the input registers from the initial two pushes and the '0xff' is the *flush_value* of all 1's for the non-pushed sub-words. Since this is a simultaneous push-flush without error and valid flush, the push causes the input register to store the `data_in` value of '0x2' in the sub-word1 position.

Figure 1-4 Push Complete Word and Push/Flush (Parameter Set 1)

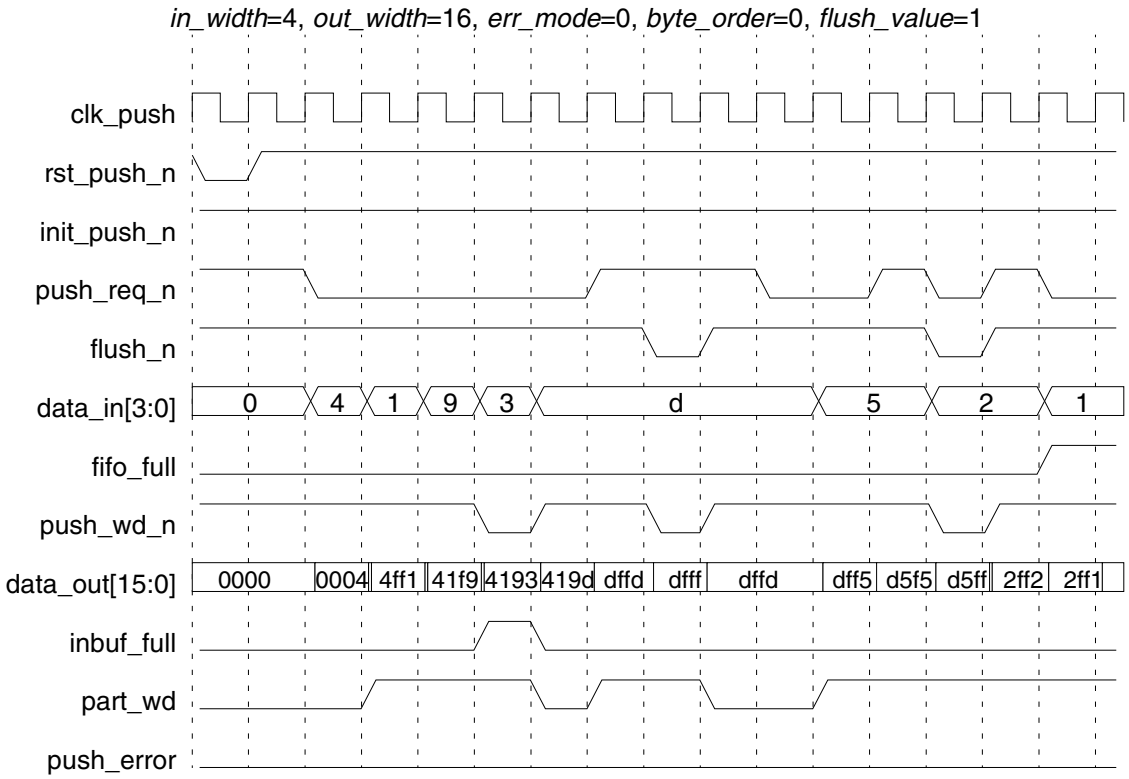


Figure 1-5 shows the same sequence as in Figure 1-4 on page 8 but with different parameter settings. In this case the 'K' is 3 (ratio of *out_width / in_width*), *byte_order* is 1, and *flush_value* is 0. So, `inbuf_full` goes active after two pushes and the `push_wd_n` active pulse occurs during the third active `push_req_n` (because `fifo_full` is not asserted). Since *byte_order* is 1, the `data_in` values pushed are placed in `data_out` starting from the least significant sub-word.

The flushing result (which occurs coincident with a push in this case) places 0's on the most significant non-pushed sub-words of `data_out` as seen by the result "0x00d" during the `push_wd_n` pulse derived from the `flush_n` assertion.

Figure 1-5 Push Complete Word and Push/Flush (Parameter Set 2)

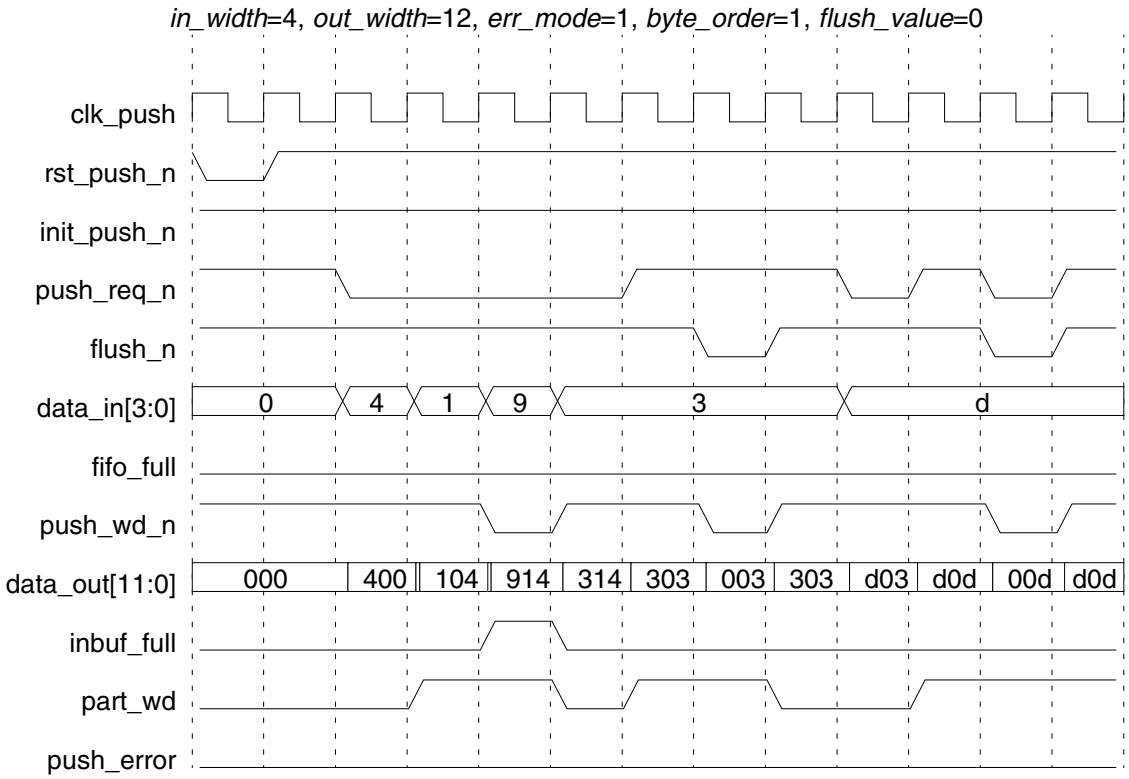
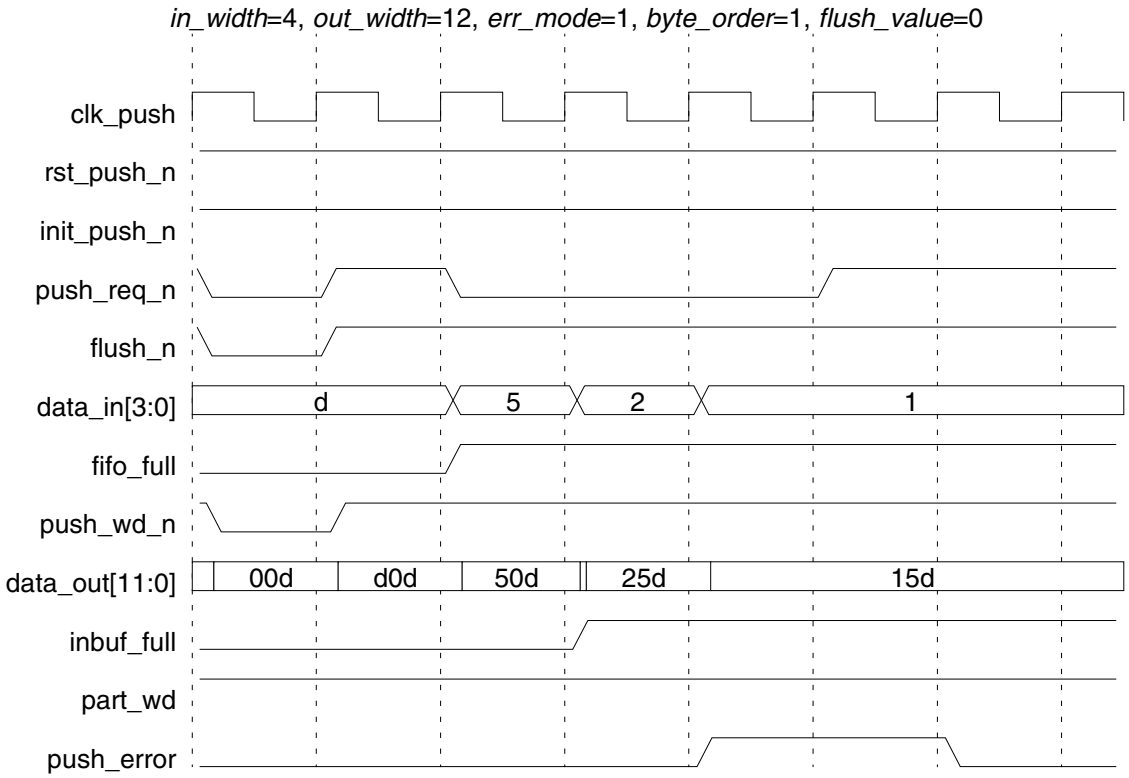


Figure 1-6 shows a push error condition. The configuration is the same as for Figure 1-5 on page 9 with the added note that *err_mode* is set to 1. This meanings that the `push_error` signal is dynamic and will report every occurrence of a push error as opposed to staying asserted on the first occurrence of an error (as when *err_mode* equals 0).

In this timing diagram a push of a complete word is attempted while `fifo_full` is asserted. That is, when `inbuf_full` and `push_req_n` are asserted while `fifo_full` is asserted. This causes a RAM/FIFO overrun error. The `push_error` is registered. The sub-word sitting at the `data_in` input for the push that caused the push error is not stored and is lost if not held constant at the input as shown in the waveform where '0x2' is dropped when `data_in` changes to '0x1' on the next clock cycle. Because *err_mode* is 1 the `push_error` assertion only lasts as longer as the error condition is present. Thus, after `push_req_n` de-asserts, `push_error` also de-asserts on the next cycle. If *err_mode* was 0, `push_error` would have remained asserted until a reset to the component was initiated.

Figure 1-6 Push Error (Parameter Set 2)



Related Topics

- Memory – FIFO Overview
- DesignWare Building Block IP User Guide

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DW_Foundation_comp.all;

entity DW_asymdata_inbuf_inst is
      generic (
        inst_in_width : INTEGER := 8;
        inst_out_width : INTEGER := 16;
        inst_err_mode : INTEGER := 0;
        inst_byte_order : INTEGER := 0;
        inst_flush_value : INTEGER := 0
        );
      port (
        inst_clk_push : in std_logic;
        inst_rst_push_n : in std_logic;
        inst_init_push_n : in std_logic;
        inst_push_req_n : in std_logic;
        inst_data_in : in std_logic_vector(inst_in_width-1 downto 0);
        inst_flush_n : in std_logic;
        inst_fifo_full : in std_logic;
        push_wd_n_inst : out std_logic;
        data_out_inst : out std_logic_vector(inst_out_width-1 downto 0);
        inbuf_full_inst : out std_logic;
        part_wd_inst : out std_logic;
        push_error_inst : out std_logic
        );
    end DW_asymdata_inbuf_inst;


architecture inst of DW_asymdata_inbuf_inst is

begin

    -- Instance of DW_asymdata_inbuf
    U1 : DW_asymdata_inbuf
    generic map ( in_width => inst_in_width, out_width => inst_out_width, err_mode =>
inst_err_mode, byte_order => inst_byte_order, flush_value => inst_flush_value )
    port map ( clk_push => inst_clk_push, rst_push_n => inst_rst_push_n, init_push_n =>
inst_init_push_n, push_req_n => inst_push_req_n, data_in => inst_data_in, flush_n =>
inst_flush_n, fifo_full => inst_fifo_full, push_wd_n => push_wd_n_inst, data_out =>
data_out_inst, inbuf_full => inbuf_full_inst, part_wd => part_wd_inst, push_error =>
push_error_inst );


end inst;

-- Configuration for use with a VHDL simulator
```

```
-- pragma translate_off
library DW03;
configuration DW_asymdata_inbuf_inst_cfg_inst of DW_asymdata_inbuf_inst is
  for inst
  end for; -- inst
end DW_asymdata_inbuf_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_asymdata_inbuf_inst( inst_clk_push, inst_rst_push_n,
     inst_init_push_n, inst_push_req_n, inst_data_in,
     inst_flush_n, inst_fifo_full, push_wd_n_inst, data_out_inst,
     inbuf_full_inst, part_wd_inst, push_error_inst );

parameter inst_in_width = 8;
parameter inst_out_width = 16;
parameter inst_err_mode = 0;
parameter inst_byte_order = 0;
parameter inst_flush_value = 0;


input inst_clk_push;
input inst_rst_push_n;
input inst_init_push_n;
input inst_push_req_n;
input [inst_in_width-1 : 0] inst_data_in;
input inst_flush_n;
input inst_fifo_full;

output push_wd_n_inst;
output [inst_out_width-1 : 0] data_out_inst;
output inbuf_full_inst;
output part_wd_inst;
output push_error_inst;

    // Instance of DW_asymdata_inbuf
    DW_asymdata_inbuf #(inst_in_width, inst_out_width, inst_err_mode, inst_byte_order,
inst_flush_value) U1 (
                .clk_push(inst_clk_push),
                .rst_push_n(inst_rst_push_n),
                .init_push_n(inst_init_push_n),
                .push_req_n(inst_push_req_n),
                .data_in(inst_data_in),
                .flush_n(inst_flush_n),
                .fifo_full(inst_fifo_full),
                .push_wd_n(push_wd_n_inst),
                .data_out(data_out_inst),
                .inbuf_full(inbuf_full_inst),
                .part_wd(part_wd_inst),
                .push_error(push_error_inst) );

endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
July 2020	DWBB_201912.5	■ Adjusted content and title of "Suppressing Warning Messages During Verilog Simulation" on page 7 and added the DW_SUPPRESS_WARN macro
October 2019	DWBB_201903.5	■ Added the "Disabling Clock Monitor Messages" section
April 2018	DWBB_201709.5	■ For STAR 9001317257, updated the maximum value for *in_width* and *out_width* parameter in Table 1-2 on page 2 ■ Added this Revision History table and the document links on this page

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com